12-1998

# Tuple Source Relational Model: A Source-Aware Data Model for Multidatabases

Ee Peng LIM
*Singapore Management University*, eplim@smu.edu.sg

Roger Hsiang-Li CHIANG

Yinyan Cao

**DOI:** https://doi.org/10.1016/S0169-023X(99)00021-X

ELSEVIER

Data & Knowledge Engineering 29 (1999) 83–114

DATA & KNOWLEDGE ENGINEERING

# Tuple source relational model: A source-aware data model for multidatabases

Ee-Peng Lim[a], Roger H.L. Chiang[b,*], Yinyan Cao[a]

[a]*Centre for Advanced Information Systems, School of Applied Science, Nanyang Technological University, Singapore 639798*

[b]*Information Management Research Centre, School of Accountancy and Business, Nanyang Technological University, Singapore 639798*

Received 7 October 1997; revised 20 March 1998; accepted 24 April 1998

**Abstract**

In some integration projects, complete integration of database instances may not be necessary. It may also be too costly and impossible to do so due to poor local data quality and insufficient instance-level knowledge. In this research, we study how multidatabases with global schemas should be represented and manipulated when the data instances from the local databases do not require to be fully integrated. We propose the tuple source (TS) relational model to represent multidatabases under such an integration requirement. This model extends the classical relational model by augmenting every relation with a source attribute to identify the local database that the tuples come from. The source attribute can also be used to specify the right context to interpret global data instances. To manipulate TS relations, we have developed a set of tuple source relational algebraic operations and an extended SQL query language known as TS-SQL. With TS relational model, flexible multidatabase queries that involve instances from different local databases can be formulated easily. In this paper, we also reported our distributed query processing and optimization strategies and their implementation. 

*Keywords:* Database integration; Multidatabases; Relational data model; Distributed query processing

## 1. Introduction

### *1.1. Background*

Databases contain data instances that represent properties of **real-world objects**. Ideally, a set of real-world objects can be described by the constructs of a single data model and stored in one and only one database. Nevertheless, in reality, one can usually find two or more databases storing

* Corresponding author. E-mail: ahlchiang@ntU.edu.sg

0169-023X/99/$ – see front matter 
PII: S0169-023X(99)00021-4

information about the same real-world objects. There are several reasons that result in the overlapping representations. These include:

- Different roles played by the same real-world objects in different applications. For example, a company can be the customer as well as the supplier for a firm. Hence, the company's information can be found in both the customers' database and suppliers' database.
- For performance reasons, a piece of information may be fully or partially duplicated and stored in databases at different geographical locations. For example, the customers' information may be stored in both the branches and the headquarter.
- Different ownership of information can also lead to information stored in different databases. For example, the information of a raw material item may be stored in different production databases because each production line wants to own a copy of the information and to exercise control over the information.

When two or more databases represent overlapping sets of real world objects, there is a strong need to integrate these databases in order to support applications of cross-functional information systems. It is therefore important to examine strategies **database integration**. An important aspect of database integration is the definition of a global schema that captures the description of the combined (or integrated) database. Here, we define **schema integration** to be the process of merging schemas of databases, and **instance integration** to be the process of integrating the database instances.

Schema integration is a problem well studied by database researchers [2,14,12,11,23]. The solution approaches identify the correspondences between schema constructs (e.g. entity types, attributes, etc.) from different databases and resolve their differences. The end result is a global schema which describes the integrated database. In contrast, instance integration focuses on merging the actual values found in instances from different databases. There are two major problems in instance integration: (a) **entity identification**; and (b) **attribute value conflict resolution**. The entity identification problem involves matching data instances that represent the same real-world objects. The attribute value conflict resolution problem involves merging the values of matching data instances. These two problems have been studied in [25,5,16] and [6,24,16,17], respectively. Note that it is not possible to have attribute value conflicts resolved without entity identification because attribute value conflict resolution can only be done for matching data instances.

In defining the integrated database, one has to choose a **global data model** so that the global schema can be described by the constructs provided by the data model. The queries that can be formulated against the integrated database also depend on the global data model. The selection of global data model depends on a number of factors including the semantic richness of the local databases [21,22] and the global application requirements. Nevertheless, the impact of instance integration on the global data model has not been well studied so far. In this paper, we study this impact in the context of relational model.

### *1.2. Objectives of work*

In this research, we assume that the schema integration process has been carried out to the extent that a global schema is obtained for a collection of existing (local) databases. Hence, global users or applications will formulate their queries based on the global schema. Moreover, export relational schemas that are compatible with respect to the global schema have been defined upon the local

databases. We classify instance integration into three distinct levels according to the extent to which instance integration is carried out:

- **Level-0.** Neither entity identification nor attribute value conflict resolution is performed. Since no instance integration is involved, the integrated database is defined merely by collecting the instances from different local databases into relations specified by the global schema.
- **Level-1.** Entity identification is performed but not attribute value conflict resolution. Hence, local database instances which correspond to the same real-world objects are matched and combined in the global relations. However, the attributes of these matching database instances are not merged.
- **Level-2 (complete integration).** Both entity identification and attribute value conflicts are resolved. In this case, the local database instances are completely integrated.

In the past database integration research, it is often thought that complete integration of instances is the only ideal solution for database integration. Nevertheless, we argue that there are some reasons advocating different levels of instance integration. Firstly, it may not be possible to acquire sufficient knowledge to perform a complete instance integration. Secondly, data quality of local databases may be low and it is not worthwhile to perform complete instance integration. Thirdly, performing instance integration can be costly, especially when the database integration is virtual and instance integration is performed for every global query. For many organizations, the benefits of complete instance integration may not outweigh costs associated with the integration. Lastly, in some cases, the global users or applications may not require a complete instance integration.

Apart from level-2 instance integration which represents the complete integration, integration levels 0 and 1 impose some constraints upon the global data model:

- Due to incomplete instance integration, the integrated database is expected to accommodate some remaining instance level heterogeneities. It is the responsibility of global applications to resolve remaining instance-level conflicts when the need arises.
- On the other hand, there exists the possibility that the levels 0 and 1 integrated databases may be needed to be fully integrated with human involvement combined with additional domain knowledge. In order to achieve this complete integration requirement, a global data model must preserve source information for partially integrated databases.
- An extended global data model associated with source information requires new set of data manipulation operations. On one hand, these operations allow us to query the integrated database. On the other hand, one can make use of these operations to achieve complete database integration.

When a complete instance integration has not been performed on multiple databases, it is necessary to augment **source** information to the global data model in order to identify where the instances in the integrated database come from. The source information further allows us to: (i) provide the context information to better interpret the non-fully integrated instances; (ii) support meaningful and flexible query formulation on the partially integrated databases; and (iii) perform entity identification and attribute value conflict resolution within queries or applications if the need arises.

In this paper, we focus only on the **level-0 instance integration** and restrict our discussion to the extension of relational model as the required global data model. We propose a **tuple source (TS) relational model** to represent multidatabases with level-0 instance integration. A corresponding set of **tuple source relational algebraic operations** is defined. We further extend the standard SQL language to allow queries on TS relations to be formulated. The research ideas and work done here can be extended to non-relational data models such as entity-relationship model and object-oriented model.

### 1.3. Outline of paper

The structure of the paper is as follows. Section 2 surveys the related research works. Section 3, present our database integration process. The proposed TS relational data model for representing multidatabases of level-0 instance integration is described in Section 4 with its relational algebraic operations. In Section 5, we discuss the unique features of TS-SQL language and their implementations. The distributed query processing architecture is given in Section 6, and our proposed query decomposition strategy is discussed in Section 7. We conclude the paper in Section 8. Appendix A discusses the formal properties of the TS data model.

## 2. Survey of related works

A number of different data models have been proposed for multidatabase systems (MDBSs). They can be broadly classified into three main categories according to the degrees of integration:

- **Type 1.** These MDBSs choose not to handle any semantic heterogeneity, e.g. MSQL [13,26,18]. In other words, they do not provide global integrated schemas over the pre-existing databases.
- **Type 2.** These MDBSs may support global integrated schemas but not integrated instances. In these MDBSs, the pre-existing database instances representing the same real-world objects are not entirely integrated together [1,15].
- **Type 3.** These are MDBSs that integrated both the pre-existing database schemas and instances [4].

The proposed **Tuple Source (TS) Relational Model** that has been designed for type 2 MDBSs. We therefore only survey a number of type 2 multidatabase systems.

In [1], a multidatabase is defined to be a set of **flexible relations** in which local instances that represent the same real-world entities are stored together as groups of tuples. Hence, some implicit grouping of tuples in a flexible relation is required. Flexible relations also capture the source, consistency and selection information of their tuples. A corresponding set of flexible relational operations has been developed to manipulate the flexible relations. Nevertheless, flexible relational model is not a natural extension of the relational model. Furthermore, the join between flexible relations has not be defined.

A universal relational approach to model and query multidatabases is proposed in [28]. In this approach, a multidatabase is a universal relation instead of a set of relations. Queries on the universal relation are translated into multiple local queries against the local relations. The final query results are formed by unioning the local query results. Source information are attached to tuples in the final query results to indicate where the tuples come from. However, the source attribute is included in neither the universal relation nor its query specification. Joins and other operations that involve multiple component databases are not allowed in this model.

## 3. Database integration process

Database integration, involving both schemas and instances of databases, should be performed in

database migration/consolidation, data warehouse, and multidatabase systems. Regardless of the mode of integration, the basic database integration tasks are essentially the same. To facilitate our discussion, we have proposed a database integration process shown in Fig. 1. We view the entire database integration as a set of processes which derives the integrated schema and instances that can be implemented on either multidatabase or data warehouse systems.

The diagram depicts the logical steps in which the integrated database is derived from the existing (local) databases. It does not dictate exactly how and when the steps should be performed. For example, for the actual consolidation of databases, schema integration and instance integration should be performed together. However, if only a virtual integration is required, schema integration will be performed once but the instance integration will be performed whenever queries are evaluated against the integrated database. The actual schema and instance integration techniques adopted will depend on a number of factors such as the global applications' requirements, types of conflicts found among local databases and data quality of local databases.

Each local database consists of a schema and a set of data instances. The schema integration process requires knowledge about the local database schemas. The knowledge about database schema

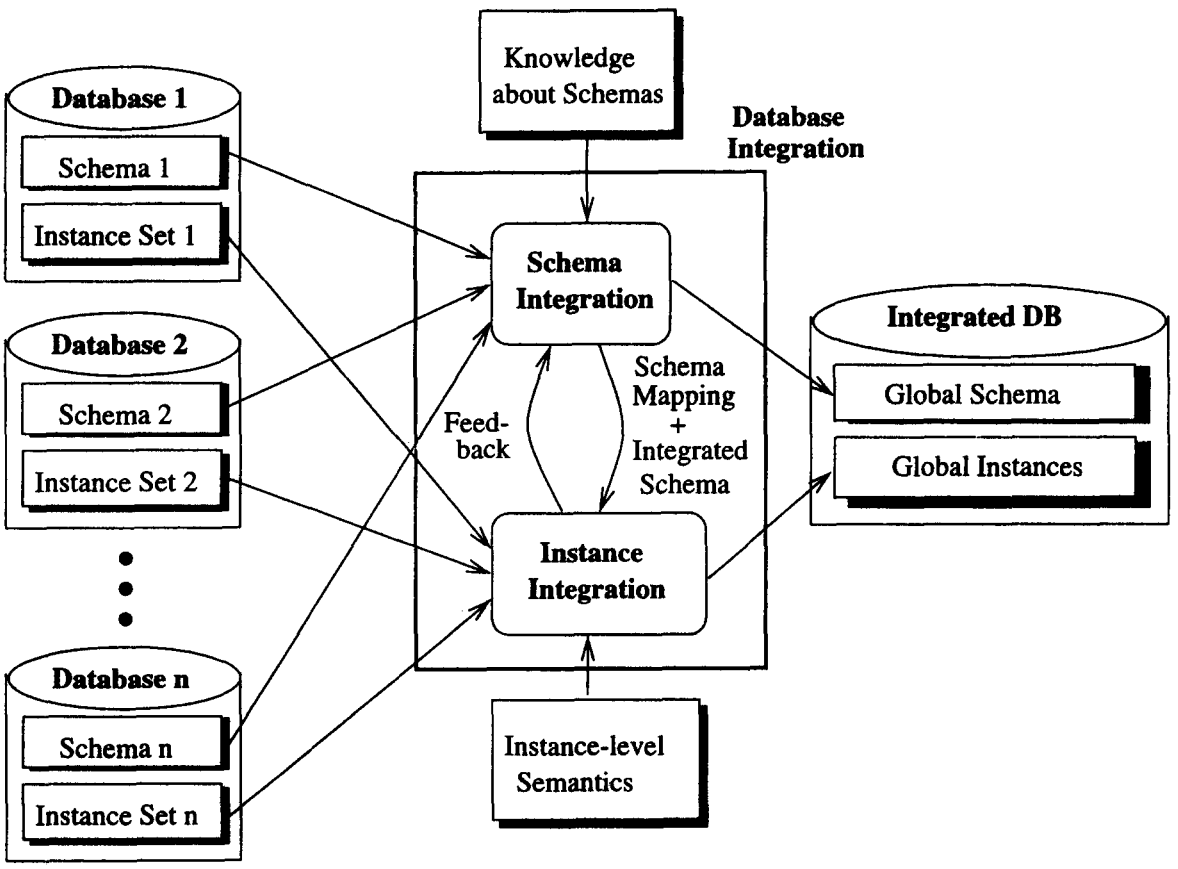



Fig. 1. Database integration process.

can be discovered from the database content. For example, database reverse engineering extracts applications' domain knowledge by analyzing not only the database schema but also database instances of an existing database [3]. However, we always require the database designers or administrators to supply additional knowledge manually. Schema integration produces the global schema as well as the mappings between the global schema elements and the local schema elements. Very often, a local schema can be vastly different from the global schema. This can be caused by different data models or database design decisions adopted by local databases and the integrated database. We may therefore have to introduce a view of the local schema, called **export schema**, such that the local database through the export schema can be seen compatible with the global schema. An export schema also defines the portion or subset of a local database to be integrated. The local database to export database conversion usually involves schema transformation. Efforts in this area are reported in [27,19,9].

The detailed instance integration process is shown in Fig. 2. Here, we show that entity identification always precedes attribute value conflict resolution since only the conflicting attribute values of matching data instances should be resolved. Throughout the entire instance integration, any detected erroneous integration result (e.g. two data instances from the same existing databases is matched to one single data instance from another database) is forwarded to the schema integration process as a feedback if the error is possibly caused by incorrect schema integration. This can happen when the schema integration makes use of hypothesis obtained by sampling the local databases. However, this hypothesis may not hold for all local database instances.

### *3.1. Example scenario*

The following are relations of two export databases which have been made schema-compatible by

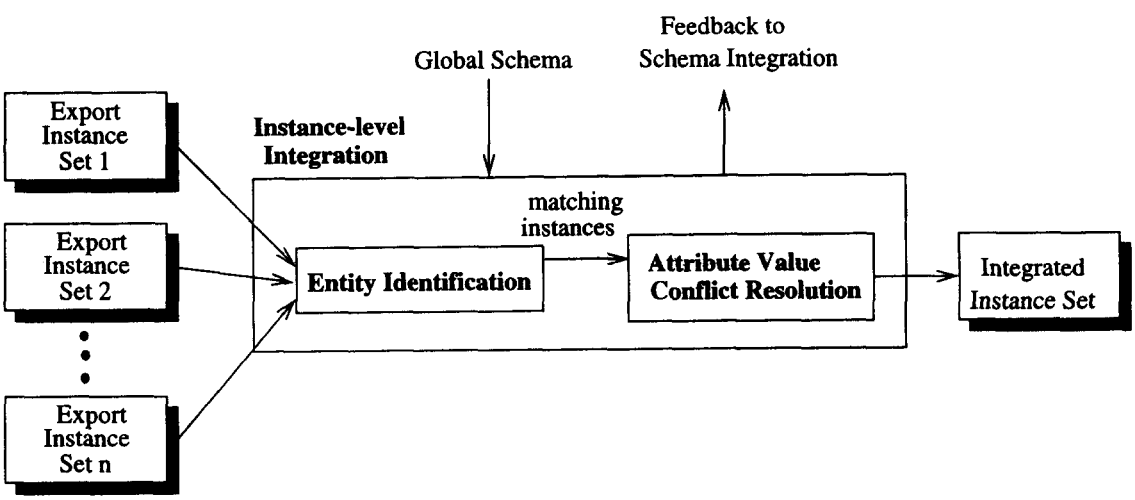



Fig. 2. Instance integration process.

the schema integration process. We will use this example to illustrate the level-0 instance integration, and TS relational model and its operations.

Export Database A
Relation $Emp_A$

| Ename | Dept | Position | Salary | Qual |
|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. |
| mark | planning | manager | $3,000 | B.Bus. |
| kim | marketing | secretary | $1,500 | NULL |
| chen | marketing | engineer | $2,500 | M.Eng. |
| daniel | library | engineer | $2,400 | B.Eng. |

Relation $Dept_A$

| Dname | Manager | Floor | Budget |
|---|---|---|---|
| marketing | chen | 3 | $2M |
| planning | mark | 2 | $4M |
| library | daniel | 1 | $1M |

Export Database B
Relation $Emp_B$

| Ename | Dept | Position | Salary |
|---|---|---|---|
| john | research | trainee | $1,200 |
| kim | marketing | secretary | $1,500 |
| chen | marketing | leader | $2,600 |
| stacy | marketing | sales rep | $3,500 |
| sugimoto | research | fellow | $10,000 |
| kain | research | engineer | $5,000 |

Relation $Dept_B$

| Dname | Manager | Floor | Budget |
|---|---|---|---|
| marketing | chan | 3 | $2.1M |
| research | sugimoto | 6 | $1M |

## 4. Tuple source relational model

At level-0 instance integration, export database instances are not integrated at all although the mapping from export schemas to global schema has been identified. It is necessary to attach the **source** information to the export instances when they appear in the global relations. This is illustrated in *Emp* and *Dept*.

As shown in *Emp*, we have assigned the ⟨*export database identifier*⟩, $DB_A$, for instances that come from the $Emp_A$ relation in Database A, and assigned $DB_B$ for instances that come from the $Emp_B$ in Database B. Since we only have one export database for each local database, the export database identifier can be treated as the local database identifier. In this way, we have extended the relational model with an additional source attribute, and we call the relational model with such extension the **Tuple Source (TS) Relational Model**.

Note that even when the schemas of our two export database examples are compatible with the global schema, there may still be global database attributes that cannot be found in all export databases. In this case, we assume NULL values for the missing attributes in the export instances. For example, all instances from $Emp_B$ do not contain the *qual* attribute and have been assigned NULL values.

Integrated Database (Level-0)
Relation *Emp*

| Ename | Dept | Position | Salary | Qual | *Source* |
|---|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. | $DB_A$ |
| john | research | trainee | $1,200 | NULL | $DB_B$ |
| mark | planning | manager | $3,000 | B.Bus. | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | $DB_B$ |
| chen | marketing | engineer | $2,500 | M.Eng. | $DB_A$ |
| chen | marketing | leader | $2,600 | NULL | $DB_B$ |
| daniel | library | engineer | $2,400 | B.Eng. | $DB_A$ |
| stacy | marketing | sales rep | $3,500 | NULL | $DB_B$ |
| sugimoto | research | fellow | $10,000 | NULL | $DB_B$ |
| kain | research | engineer | $5,000 | NULL | $DB_B$ |

Relation *Dept*

| Dname | Manager | Floor | Budget | *Source* |
|---|---|---|---|---|
| marketing | chen | 3 | $2M | $DB_A$ |
| marketing | chan | 3 | $2.1M | $DB_B$ |
| planning | mark | 2 | $4M | $DB_A$ |
| library | daniel | 1 | $1M | $DB_A$ |
| research | sugimoto | 6 | $1M | $DB_B$ |

At first glance, one may want to treat the additional source attribute like just another normal attribute. While this may be correct at the data storage level, we advocate that the source attribute deserves special treatment at both the data modeling and the query processing perspectives. The source attribute, unlike other normal attributes, must be present in every TS relation and has a special meaning which not only relates the instances to the local databases they come from, but also identify the context of data instances. Furthermore, it should be manipulated differently from the other normal attributes in the query processing (see Section 4.2).

For the TS relational model, the values of source attributes are used purely for implementation purpose. They do not provide any semantics regarding local databases. In order to maintain and provide source (context) semantics in a multidatabase, we can establish a source table with at least two attributes. The first attribute stores the local database identifiers, whereas the other attributes store information about the local databases. Information could be the application domains, the names of geographical locations, the types of database management systems, persons in charge (e.g., DBA), and even the assessment of the data quality's level of each local database. This source table is employed to retain the context semantics which can be inferred by users to interpret global queries' results of multidatabase with level-0 instance integration, or used by other data analysis tools. In addition, this table contains useful information for level-1 instance integration. For example, the context semantics of our example can be stored in the following source table.

Source Relation

| *source-id* | **org-unit** | **location** | **DBMS** | **DBA** |
|---|---|---|---|---|
| $DB_A$ | Personnel office | Albany, New York, USA | DB2 | Franklin Wong |
| $DB_B$ | Payroll department | Rochester, New York, USA | ORACLE7 | Jennifer Wallace |

Being an extension to the traditional relational model, TS relational model can represent relations which do not require source information by assigning * values to the source attributes. The standard relational operations can still operate on the TS relations by ignoring the source attributes. Note that the resultant relations may no longer retain the values of the source attributes. With the special meaning attached to the source attribute, we design manipulation operations that involve the source attributes, which are called **tuple source (TS) relational algebraic operators**.

### 4.1. Merge operation

In deriving level-0 global relations from the export relations, we assume that each global relation is formed by one or none export relation from each export database. This assumption is reasonable since the export database should have a schema compatible with that of integrated database. To combine the export relations of a global relation, we need a **merge operation**.

Let $DB_1, \ldots, DB_n$ be $n$ export databases, and let $L_{ij}$ represent a relation in $DB_i$ which is a component of the global relation $G_j$. $G_j$ is derived from the component export relations $L_{1j}$, $L_{2j}, \ldots, L_{nj}$ by:

$$G_j = merge(L_{1j}, L_{2j}, \ldots, L_{nj})$$

**Definition.** (*merge*)
Let $\mathscr{DB}$ be the set of all export databases, $DB_1, \ldots, DB_n$. Let $L_{1j}, \ldots, L_{nj}$ be export relations from $DB_1, \ldots, DB_n$, respectively. They all correspond to the same the global relation $G_j$. $L_{ij}$ is empty if no export relation from $DB_i$ corresponds to $G_j$. Let $A = Attr(L_{1j}) \cup \cdots \cup Attr(L_{nj})$.

$merge(L_{1j}, \ldots, L_{nj}) = extend(L_{1j}, A, `DB_1') \cup \cdots \cup extend(L_{nj}, A, `DB_n')$
where $extend(L_{ij}, A, `DB_j')$ augments records in $L_{ij}$ with NULL values for attributes in $A$ which are not found in $L_{ij}$, and $`DB_j'$ for the *source* attribute.

In some way, the *merge* operation is similar to an outer-union operation except that an additional source attribute is added to each of the relations before they are outer-unioned.

**Example.** The global relations *Emp* and *Dept* can be derived by:
$Emp = merge(Emp_A, Emp_B)$
$Dept = merge(Dept_A, Dept_B)$

### 4.2. Tuple source (TS) relational operations

In this section, we introduce a new set of operators for TS relations. These include TS select($\overset{s}{\sigma}$), project($\overset{s}{\pi}$), join($\overset{s}{\bowtie}$), minus($\overset{s}{-}$), union ($\overset{s}{\cup}$), intersect($\overset{s}{\cap}$), aggregate($\overset{s}{aggreg}$), and groupby($\overset{s}{groupby}$) operations. To differentiate the TS relational operators from the traditional relational operators, we add $s$ to the operator symbols. The following are formal definitions of these operators:

**Definition (TS select operation—$\overset{s}{\sigma}$).** Let $R$ be a TS relation. We use $(t, s)$ to denote a tuple in $R$ where $t$ represents the normal attribute values and $s$ represents the source attribute value. The TS select operation is defined as:

$$\overset{s}{\sigma}_{pred} R = \{(t, s) | (t, s) \in R \wedge pred(t, s)\}$$

Here, $pred(t, s)$ is a predicate that can involve both normal and source attributes. Currently, we have $pred(t, s)$ comprises of either $(source \in DBSet)$ or $(source = *)$ where *DBSet* is a set of export database identifiers. We call the predicates on source attributes the **source predicates**. By its definition, $\overset{s}{\sigma}$ does not modify the source attribute values of its operand relation.

**Example.** To retrieve the information of employees who work in the marketing department from $DB_A$, the following $\overset{s}{\sigma}$ operation can be performed.

$\overset{s}{\sigma}_{(dept='marketing')\wedge(source\in\{DB_A\})}Emp$

| Ename | Dept | Position | Salary | Qual | *Source* |
|---|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | $DB_A$ |
| chen | marketing | engineer | $2,500 | M.Eng. | $DB_A$ |

**Example.** To retrieve the information of employees who are trainees, the following $\overset{s}{\sigma}$ operation can be performed.

$\overset{s}{\sigma}_{(position='trainee')}Emp$

| Ename | Dept | Position | Salary | Qual | *Source* |
|---|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. | $DB_A$ |
| john | research | trainee | $1,200 | NULL | $DB_B$ |

**Definition (TS project operation—$\overset{s}{\pi}$).** Let $R$ be a TS relation, and $A$ be a subset of normal attributes in $R$. The TS project operation is defined as:

$\overset{s}{\pi}_{A}^{same\_DB}R = \pi_{(A,source)}R$

$\overset{s}{\pi}_{A}^{any}R = \{(t.A, source\_merge(\pi_{source}\sigma_{A=t.A}R))|t \in R\}$

where

$$source\_merge(S) = \begin{cases} s & \text{if } s \in S \text{ and all members of } S \text{ are identical} \\ & \text{where } S \text{ is a set of source values} \\ * & \text{otherwise} \end{cases}$$

Unlike the normal projection, we attach a flag (*same_DB* or *any*) to $\overset{s}{\pi}$ to indicate if projected tuples from different export databases sharing the same normal attribute values should be merged or not. $\overset{s}{\pi}^{same\_DB}$ does not modify the source attribute values of its operand relation. In contrast, $\overset{s}{\pi}^{any}$ indicates the source statuses of tuples sharing the same projected attribute values.

Note that source_merge() produces * for those resultant tuples having multiple sources. It does not maintain the original sources for a number of reasons. Firstly, source attribute values should be atomic. Secondly, even if we maintain set values for source information, it is still not possible to tell the exact source of each individual attribute for a given TS relation. Due to TS project and TS join operations, the source value of a tuple could be derived from either merging of export tuples from different sources or from inheriting attributes from different sources.

**Example.**

$\overset{s}{\pi}_{dname,manager}^{same\_DB}Dept$

| Dname | Manager | *Source* |
|---|---|---|
| marketing | chen | $DB_A$ |
| marketing | chan | $DB_B$ |
| planning | mark | $DB_A$ |
| library | daniel | $DB_A$ |
| research | sugimoto | $DB_B$ |

$\overset{s}{\pi}_{dname}^{any}Dept$

| Dname | *Source* |
|---|---|
| marketing | * |
| planning | $DB_A$ |
| library | $DB_A$ |
| research | $DB_B$ |

**Definition (TS join operation—$\overset{s}{\bowtie}$).** Let $R$ and $S$ be two TS relations.

$$R \overset{s}{\bowtie}_{pred}^{same\_DB} S = \{(a_r, a_s, s_r)|(a_r, s_r) \in R \wedge (a_s, s_s) \in S \wedge pred(a_r, a_s, s_r, s_s) \wedge (s_r = s_s \neq *)\}$$

$$R \overset{s}{\bowtie}_{pred}^{any} S = \{(a_r, a_s, source\_merge(\{s_r, s_s\}))|(a_r, s_r) \in R \wedge (a_s, s_s) \in S \wedge pred(a_r, a_s, s_r, s_s)\}$$

where *pred* is a conjunction of predicates which may include the following source-related predicates: $(s_r = db\_id)$, $(s_s = db\_id)$, $(s_r = *)$ and $(s_s = *)$.

Unlike $\overset{s}{\bowtie}^{same\_DB}$ which retains the original non-* source values, $\overset{s}{\bowtie}^{any}$ generates new source values for the result tuples according to the *source_merge*() function. In particular, $\overset{s}{\bowtie}^{any}$ will merge two different source values into *.

**Example.** If we want to obtain employees and their department information together based on their associations within the local databases, the following $\overset{s}{\bowtie}$ can be performed.

$Emp \overset{s}{\bowtie}^{same\_DB}_{dept=dname} Dept$

| Ename | Dept | Position | Salary | Qual | Dname | Manager | Floor | Budget | *Source* |
|---|---|---|---|---|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. | marketing | chen | 3 | $2M | $DB_A$ |
| john | research | trainee | $1,200 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |
| mark | planning | manager | $3,000 | B.Bus. | planning | mark | 2 | $4M | $DB_A$ |
| kim | marketing | sectetary | $1,500 | NULL | marketing | chen | 3 | $2M | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| chen | marketing | engineer | $2,500 | M.Eng. | marketing | chen | 3 | $2M | $DB_A$ |
| chen | marketing | leader | $2,600 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| daniel | library | engineer | $2,400 | B.Eng. | library | daniel | 1 | $1M | $DB_A$ |
| stacy | marketing | sales rep | $3,500 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| sugimoto | research | fellow | $10,000 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |
| kain | research | engineer | $5,000 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |

**Example.** To retrieve employees and their department information together regardless of the local databases the data come from, the following $\overset{s}{\bowtie}$ can be performed.

$Emp \overset{s}{\bowtie}^{any}_{dept=dname} Dept$

| Ename | Dept | Position | Salary | Qual | Dname | Manager | Floor | Budget | *Source* |
|---|---|---|---|---|---|---|---|---|---|
| john | marketing | trainee | $1,000 | Dipl. | marketing | chen | 3 | $2M | $DB_A$ |
| john | marketing | trainee | $1,000 | Dipl. | marketing | chan | 3 | $2.1M | * |
| john | research | trainee | $1,200 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |
| mark | planning | manager | $3,000 | B.Bus. | planning | mark | 2 | $4M | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | marketing | chen | 3 | $2M | $DB_A$ |
| kim | marketing | secretary | $1,500 | NULL | marketing | chan | 3 | $2.1M | * |
| kim | marketing | secretary | $1,500 | NULL | marketing | chen | 3 | $2M | * |
| kim | marketing | secretary | $1,500 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| chen | marketing | engineer | $2,500 | M.Eng. | marketing | chen | 3 | $2M | $DB_A$ |
| chen | marketing | engineer | $2,500 | M.Eng. | marketing | chan | 3 | $2.1M | * |
| chen | marketing | leader | $2,600 | NULL | marketing | chen | 3 | $2M | * |
| chen | marketing | leader | $2,600 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| daniel | library | engineer | $2,400 | B.Eng. | library | daniel | 1 | $1M | $DB_A$ |
| stacy | marketing | sales rep | $3,500 | NULL | marketing | chen | 3 | $2M | * |
| stacy | marketing | sales rep | $3,500 | NULL | marketing | chan | 3 | $2.1M | $DB_B$ |
| sugimoto | research | fellow | $10,000 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |
| kain | research | engineer | $5,000 | NULL | research | sugimoto | 6 | $1M | $DB_B$ |

**Definition (TS union operation—$\overset{s}{\cup}$).** Let $R$ and $S$ be two TS relations, and $A$ be their common set of attributes.

$$R \overset{s}{\cup}^{same\_DB} S = R \cup S$$

$$R \overset{s}{\cup}^{any} S = \{(t.A, source\_merge(\pi_{source}(\sigma_{A=t.A}(R \cup S))))|t.A \in (\pi_A R \cup \pi_A S)\}$$

**Definition (TS intersect operation—$\overset{s}{\cap}$).** Let $R$ and $S$ be two TS relations.

$$R \overset{s}{\cap}^{same\_DB} S = R \cap S$$

$$R \overset{s}{\cap}^{any} S = \{(t.A, source\_merge(\pi_{source}(\sigma_{A=t.A}(R \cap S))))|t.A \in (\pi_A R \cap \pi_A S)\}$$

**Definition (TS minus operation—$\overset{s}{-}$).** Let $R$ and $S$ be two relations.

$$R \overset{s}{-}^{same\_DB} S = R \cap S$$

$$R \overset{s}{-}^{any} S = \{(t.A, source\_merge(\pi_{source}(\sigma_{A=t.A}(R \cup S))))|t.A \in (\pi_A R - \pi_A S)\}$$

The above three definitions indicate that a TS set operation with *any* flag (e.g. $\overset{s}{-}^{any}$) is performed by first performing the corresponding normal set operation (e.g. $-$) of normal attributes on the TS relations followed by deriving the source values of the result tuples.

**Example.** Let $T_1$ and $T_2$ be two TS relations as shown below. The various source aware union, intersect and minus operations can be performed as follows:

*Relation* $T_1$

| Ename | Dept | *Source* |
|---|---|---|
| john | research | $DB_A$ |
| john | research | $DB_B$ |
| tom | sales | $DB_A$ |
| peter | accounting | $DB_A$ |
| mary | sales | $DB_B$ |

Relation $T_2$

| Ename | Dept | *Source* |
|---|---|---|
| tom | sales | $DB_A$ |
| mary | sales | $DB_A$ |
| mark | marketing | $DB_B$ |

$T_1 \overset{s}{\cup}^{same\_DB} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| john | research | $DB_A$ |
| john | research | $DB_B$ |
| tom | sales | $DB_A$ |
| peter | accounting | $DB_A$ |
| mary | sales | $DB_B$ |
| mary | sales | $DB_A$ |
| mark | marketing | $DB_B$ |

$T_1 \overset{s}{\cup}^{any} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| john | research | * |
| tom | sales | $DB_A$ |
| peter | accounting | $DB_A$ |
| mary | sales | * |
| mark | marketing | $DB_B$ |

$T_1 \overset{s}{\cap}^{same\_DB} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| tom | sales | $DB_A$ |

$T_1 \overset{s}{\cap}^{any} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| tom | sales | $DB_A$ |
| mary | sales | * |

$T_1 \overset{s}{-}{}^{same\_DB} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| john | research | $DB_A$ |
| john | research | $DB_B$ |
| peter | accounting | $DB_A$ |
| mary | sales | $DB_B$ |

$T_1 \overset{s}{-}{}^{any} T_2$

| Ename | Dept | *Source* |
|---|---|---|
| john | research | * |
| peter | accounting | $DB_A$ |

Here, we observe that $(R \overset{s}{\cap}{}^{same\_DB} S) \subseteq (R \overset{s}{\cup}{}^{same\_DB} S)$. The same subset relationship also exists between $\overset{s}{\cap}{}^{any}$ and $\overset{s}{\cup}{}^{any}$.

An important goal for database integration is to allow global applications to collect statistics about the local databases. In particular, organization-wide information systems usually require summaries of local information in order to assist managers in making decisions. In the following, we define the TS version of aggregation and group by operations.

**Definition (TS aggregate operation—*$\overset{s}{aggreg}$*).** Let $R$ be a TS relation. Let $A_1, \ldots, A_m$ be $R$'s normal attributes, and $f_1, \ldots, f_m$ be functions each of which operates on a set of values.

$$\overset{s}{aggreg}{}^{any} R(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle) = \{(f_1(\pi_{A_1} R), \ldots, f_m(\pi_m R), source\_merge(\pi_{source} R))\}$$

$$\overset{s}{aggreg}{}^{same\_DB} R(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle) = \{(f_1(\pi_{A_1}(\sigma_{source=DB_i} R)), \ldots, f_m(\pi_{A_m}(\sigma_{source=DB_i} R)), DB_i) | DB_i \in \pi_{source} R\} \cup \{(f_1(\pi_{A_1}(\sigma_{source=*} R)), \ldots, f_m(\pi_{A_m}(\sigma_{source=*} R)), *)\}$$

**Example.** To find the total budgets for departments from different databases, the following $\overset{s}{aggreg}$ operation can be performed:

$\overset{s}{aggreg}{}^{same\_DB} Dept(\langle sum, budget\rangle)$

| Sum(budget) | *Source* |
|---|---|
| \$7M | $DB_A$ |
| \$3.1M | $DB_B$ |

**Definition (TS group-by operation—*$\overset{s}{groupby}$*).** Let $R$ be a TS relation. Let $A_1, \ldots, A_m$ be $R$'s normal attributes, $B$ be the group-by attribute, and $f_1, \ldots, f_m$ be functions.

$$\overset{s}{groupby}{}^{any} R(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B) = \{(f_1(\pi_{A_1}(\sigma_{B=bi} R)), \ldots, f_m(\pi_{Am}(\sigma_{B=bi} R)), source\_merge(\pi_{source}(\sigma_{B=bi} R))) | b_i \in \pi_B R\}$$

$$\overset{s}{groupby}{}^{same\_DB} R(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B) = \{(f_1(\pi_{A_1}(\sigma_{(B=bi)\wedge(source=DB_i)} R)), \ldots, f_m(\pi_{A_m}(\sigma_{(B=bi)\wedge(source=DB_i)} R)), b_i, DB_i) | (b_i \in \pi_B R) \wedge (DB_i \in \pi_{source} R)\} \cup (f_1(\pi_{A_1}(\sigma_{(B=bi)\wedge(source=*)} R)), \ldots, f_m(\pi_{A_m}(\sigma_{(B=bi)\wedge(source=*)} R)), b_i, *) | b_i \in \pi_B R\}$$

**Example.** To find the total salaries for employees grouped by departments from different databases, the following *groupby* operation can be performed:

$\overset{s}{groupby}^{same_{DB}}Emp(\langle sum, salary\rangle)(dept)$

| Sum(salary) | Dept | *Source* |
|---|---|---|
| $5,000 | marketing | $DB_A$ |
| $3,000 | planning | $DB_A$ |
| $2,400 | library | $DB_A$ |
| $16,200 | research | $DB_B$ |
| $7,600 | marketing | $DB_B$ |

For $\overset{s}{aggreg}^{any}$ and $\overset{s}{groupby}^{any}$ operations, it is usually not advisable to use functions such as *count* and *sum* since multiple tuples in a TS relation may actually model the same real-world entities.

## 5. Tuple source-structured query language (TS-SQL)

### *5.1. Query syntax*

Like the normal relational operations, TS relational operations have well defined semantics but they may not be suitable for query formulation. To facilitate query formulation, we design the tuple source version of SQL, known as TS-SQL. A simple TS-SQL query demonstrates the following syntax:

```
select ⟨target_attributes⟩ [any]/[same_DB]
with source context (optional)
from ⟨TS_relations⟩
where ⟨selection_join_conditions⟩ [any] / [same_DB]
```

As shown above, simple TS-SQL queries looks very much like normal SQL queries except that an optional keyword of *any* or *same_DB* can be added to the `select` clause and `where` clause, and another optional `with` clause. The keyword added to the `select` clause indicates if the TS-projection is to be performed with the *any* or *same_DB* flag. The keyword added to the `where` clause indicates the flag to be used in a TS-join. For both clauses, the default keyword used is *any*. ⟨*selection_join_conditions*⟩ can also include source predicate(s), which is of the form ⟨*TS_relation*⟩ *.source = DB_id*⟩. Unlike normal SQL queries, TS-SQL query results always include the source attribute even when it is not specified as the target attributes. By specifying the `with` clause in a query, the query result will also include the source context allowing the user to interpret the query result tuples using the source context.

Given the following TS-SQL query,

```
select R.A, S.B, T.C [same_DB]
from R, S, T
where R.X=S.X and S.Y=T.Y and R.Z=T.Z and
  R.U='ABC' and S.V='DEF' [any]
```

We can translate it into the equivalent TS relational expression shown below:

$$\overset{s}{\pi}{}^{same\_DB}_{R.A,S.B,T.C}((\overset{s}{\sigma}_{R.U='ABC'}R) \overset{s}{\bowtie}{}^{any}_{R.X=S.X} (\overset{s}{\sigma}_{S.V='DEF'}S) \overset{s}{\bowtie}{}^{any}_{(S.Y=T.Y)\wedge(R.Z=T.Z)} T)$$

The translation is possible because both $\overset{s}{\bowtie}{}^{any}$ and $\overset{s}{\bowtie}{}^{same\_DB}$ are commutative and associative. Since source attributes of operand relations are merged during the join, source predicates must be evaluated before any TS join is carried out.

A TS-SQL query involving union, intersection, or subtraction of two or more select queries can be written as:

`select` ⟨*target_attributes*⟩ `[any] / [same_DB]`
`from` ⟨*TS_relations*⟩
`where` ⟨*selection_join_conditions*⟩ `[any] / [same_DB]`
`union / intersect / minus` [*any or same_DB*]
`select` ⟨*target_attributes*⟩ `[any] / [same_DB]`
`from` ⟨*TS_relations*⟩
`where` ⟨*selection_join_conditions*⟩ `[any] / [same_DB]`

Aggregate and groupby TS-SQL queries are similar to their SQL counterparts as shown below:

`select f1` (⟨*target_attribute*$_1$⟩), . . . , `fn`(⟨*target_attribute*$_n$⟩) `[any] / [same_DB]`
`from` ⟨*TS_relations*⟩
`select f1` (⟨*target_attribute*$_1$⟩), . . . , `fn`(⟨*target_attribute*$_n$⟩) `[any] / [same_DB]`
`from` ⟨*TS_relations*⟩ `groupby` ⟨*groupby_attribute*⟩

Here, the keyword assigned to the `select` clause indicates the flag to be used in TS aggregate and groupby operations. In summary, the main features offered by TS-SQL include:

- TS-SQL satisfies the *closure property*. Given TS-relations, TS-SQL queries produce TS-relations as results.
- Unlike the traditional SQL, TS-SQL allows source options (*SAME_DB* and *ANY_DB*) to be specified on the SELECT and WHERE clauses, as well as on the union and intersect operations. These options dictate how the tuples in the TS-relations are processed based on their source values. Source option specified on a SELECT clause determines if tuples from different local databases can be combined during projection. Source option specified on a WHERE clause determines if tuples from different local databases can be combined during join.
- Queries to specific local databases can be formulated by defining source predicates to operand TS-relations. A source predicate is represented by ⟨*relation_name*⟩*.source in* ⟨*set_of_local_DB_IDs*⟩.
- Aggregation and groupby operations are customized to handle summarization of tuples based on their source values.

Despite TS-SQL resembles the traditional SQL closely, it can be shown that some of the TS-SQL queries involving the ANY_DB option cannot be performed both directly and indirectly by the normal SQL. Even when some of the TS-SQL queries can be computed by SQL expressions, we believe that TS-SQL will greatly reduce the effort of query formulation for multidatabses with level-0 instance integration.

### *5.2. Simple query examples*

Simple TS-SQL queries are SELECT-PROJECT-JOIN queries. In the following, we show a number of simple TS-SQL queries and explain their semantics.

**Example (Q1).** Retrieve the name, salary and qualification of employees with salary less than 3000 with reference to the local databases.

```
SELECT E1.ename, E1.salary, E1.qual [SAME_DB] FROM Emp E1 WHERE E1.salary <3000
```

Q1's result

| E1.ename | E1.salary | E1.qual | *Source* |
|---|---|---|---|
| john | $1,000 | Dipl. | $DB_A$ |
| john | $1,200 | NULL | $DB_B$ |
| kim | $1,500 | NULL | $DB_A$ |
| kim | $1,500 | NULL | $DB_B$ |
| chen | $2,500 | M.Eng. | $DB_A$ |
| chen | $2,600 | NULL | $DB_B$ |
| daniel | $2,400 | B.Eng. | $DB_A$ |

With the source option *SAME_DB* assigned to the SELECT clause, Q1 requires the projection of *Emp* table to include the source attribute. Hence, tuples with identical projected attribute values but not source values remain to be separate in the query result, e.g. the information about *kim*. If the source information is not important during projection, the source option *ANY_DB* can be assigned to the SELECT clause as shown in Q2 below.

**Example (Q2).** Retrieve the different positions held by employees regardless where the employee records come from.

```
SELECT E1.position [ANY_DB] FROM Emp E1
```

Q2's result

| E1.position | *Source* |
|---|---|
| trainee | * |
| manager | $DB_A$ |
| secretary | * |
| engineer | * |
| leader | $DB_B$ |
| sales rep | $DB_B$ |
| fellow | $DB_B$ |

As shown in Q2's result, positions that can be found in both local databases have * as their source values. The * value indicates that a tuple has been obtained by merging tuples from different local databases. In this case, we lose the source information of such tuples.

If the user would like to view the source context together withthe result tuples, he/she can specify the `with` clause in the query as shown in Q3 below. Note that the addition of `with` clause causes the source relation to be joined with the TS relation(s) in the `where` clause using the source attribute.

When a tuple has * as the source value, its source context will carry NULL values for the context attributes.

**Example (Q3).** Retrieve the positions held by employees regardless of where the employee records come from and include the source context information.

```
SELECT E1.position [ANY_DB] WITH SOURCE CONTEXT FROM Emp E1
```

Q3's result

| E1.position | *Source* | Org_unit | Location | DBMS | DBA |
|---|---|---|---|---|---|
| trainee | * | NULL | NULL | NULL | NULL |
| manager | $DB_A$ | Personnel office | Albany, New York, USA | DB2 | Franklin Wong |
| secretary | * | NULL | NULL | NULL | NULL |
| engineer | * | NULL | NULL | NULL | NULL |
| leader | $DB_B$ | Payroll department | Rochester, New York, USA | ORACLE7 | Jennifer Wallace |
| sales rep | $DB_B$ | Payroll department | Rochester, New York, USA | ORACLE7 | Jennifer Wallace |
| fellow | $DB_B$ | Payroll department | Rochester, New York, USA | ORACLE7 | Jennifer Wallace |

When two or more relations are given in the WHERE clause of TS-SQL, a source option can be assigned to the WHERE clause to indicate how the relations are to be joined together with respect to their source attributes.

**Example (Q4).** Retrieve the employees and their managers according to the local databases they come from.

```
SELECT E1.ename, D1.manager [SAME_DB] FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname [SAME_DB]
```

Q4's result

| E1.ename | D1.manager | *Source* |
|---|---|---|
| john | chen | $DB_A$ |
| john | sugimoto | $DB_B$ |
| mark | mark | $DB_A$ |
| kim | chen | $DB_A$ |
| kim | chan | $DB_B$ |
| chen | chen | $DB_A$ |
| chen | chan | $DB_B$ |
| daniel | daniel | $DB_A$ |
| stacy | chan | $DB_B$ |
| sugimoto | sugimoto | $DB_B$ |
| kain | sugimoto | $DB_B$ |

**Example (Q5).** Retrieve the employees and their managers regardless of the local databases they come from.

```
SELECT E1.ename, D1.manager [ANY_DB] FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname [ANY_DB]
```

By disregarding the source information, Q5 allows us to establish any possible relationship between the employees and managers across the local databases.

The source attribute can further allow us to formulate queries that retrieve tuples from specific local database(s) as shown in Q6.

**Example (Q6).** Retrieve the employees and their managers from local database $DB_A$.[1]

```
SELECT E1.ename, D1.manager [SAME_DB] FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname and *.source in {DB_A} [SAME_DB]
```

The predicate (*.source in $\{DB_A\}$) abbreviates ((E1.source in $\{DB_A\}$) and (D1.sources in $\{DB_A\}$)). In Q6, as all tuples to be joined come from the $DB_A$, the source options assigned to SELECT and WHERE clauses can be ignored.

### 5.3. Aggregate and groupby queries

TS-SQL also supports aggregate and groupby queries. As shown in the following query examples, by assigning different source options to the SELECT clauses, we can obtain summarized information with or without reference to the source attributes.

**Example (Q7).** Calculate the average salary for all employees in the Emp relation regardless of where they come from.

```
SELECT avg(E1.salary) [ANY_DB] FROM Emp E1
```

Q7's result

| avg(E1.salary) | *Source* |
|---|---|
| 3109 | * |

**Example (Q8).** Calculate, for each department from each local database, the number of employees earning more than $2000.

```
SELECT count(*), E1.dept [SAME_DB] FROM Emp E1
WHERE E1.salary > 2,000 GROUPBY E1.dept
```

Q8's result

| Count(*) | E1.dept | *Source* |
|---|---|---|
| 1 | planning | $DB_A$ |
| 1 | marketing | $DB_A$ |
| 2 | marketing | $DB_B$ |
| 1 | library | $DB_A$ |
| 2 | *research* | $DB_B$ |

[1] To conserve space, we do not show the result here.

## 6. Distributed query processing architecture

### 6.1. Query mediator and query agent

As shown in Fig. 3, our distributed query processor consists of a **query mediator** and a number of **query agents**, one for each local database. The query mediator is responsible for decomposing global queries given by multidatabase applications into multiple subqueries to be evaluated by the query agents. It also assembles the subquery results returned by the query agents and further processes the assembled results in order to compute the final query result. Query agents transform subqueries into local queries that can be directly processed by the local database systems. The local query results are properly formatted before they are forwarded to the query mediator.

By dividing the query processing tasks between query mediator and query agents, we allow concurrent processing of subqueries on local databases located at different sites, thus reducing the query response time. This architectural design further enables the query mediator to focus on global query processing and optimization while the query agents handle the transformation of subqueries decomposed by query mediator into local queries. Note that the query decomposition performed by query mediator assumes all local database schemas are compatible with the global schema. It is the job of query agents to convert the subqueries into local queries on possibly heterogeneous local schemas. The heterogeneous query interfaces of local database systems are also hidden from the query mediator by the query agents.

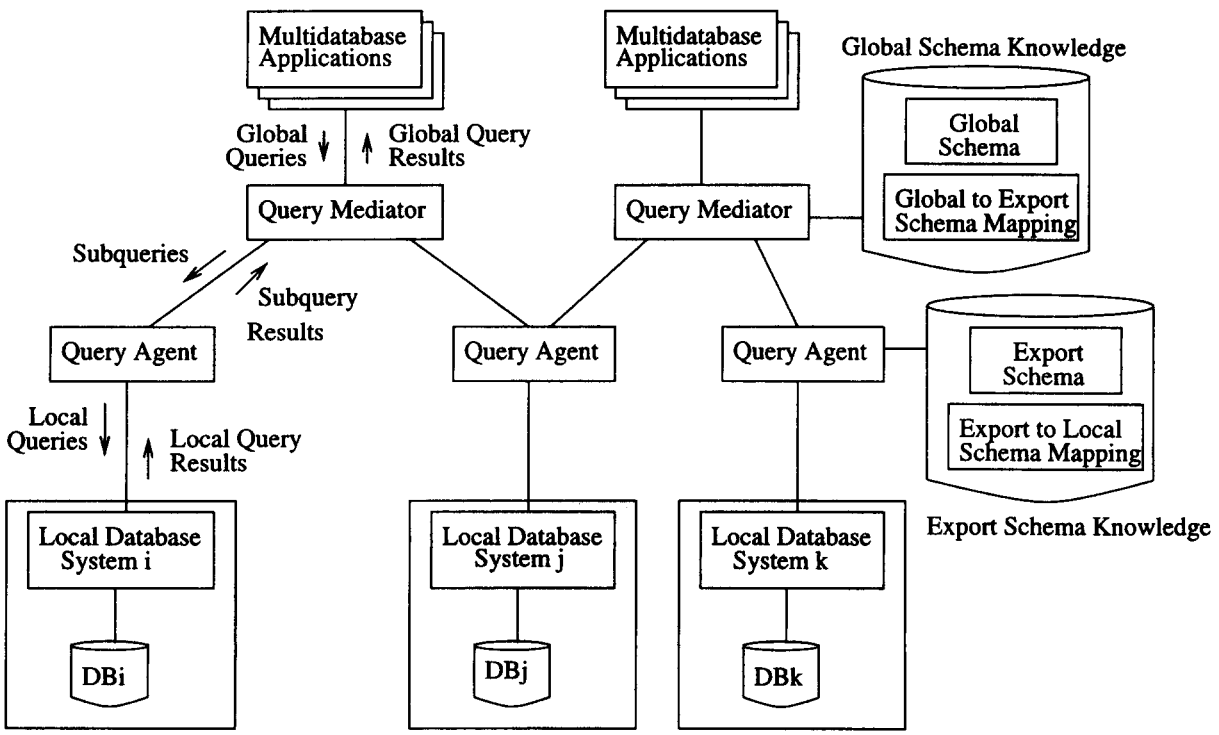



Fig. 3. Multidatabase query processing architecture.

The query mediator and query agents require different types of schema knowledge for their query processing tasks. To decompose global queries into subqueries, the query mediator requires the global schema and global schema to export schema mapping knowledge. We define export schema to be the schema that is supported by a query agent. The export schema adopted by a query agent must be compatible with the global schema adopted by its query mediator. Similarly, the export schema and export to local schema mapping knowledge are information required by query agents to perform their query transformation.

### *6.2. Distributed query processing steps*

The overall distributed query processing steps designed for global TS-SQL queries are shown in Fig. 4. We briefly describe these steps as follows:

(1) **Query parsing.** Global TS-SQL queries are parsed to ensure that they are syntactically correct. Based on the parsed trees constructed, the queries are validated against the global schema to ensure that all relations and attributes in the queries exist and are properly used.

(2) **Query decomposition.** Given a global TS-SQL query, we decompose it into subqueries to be evaluated by the query agents. Here, the local databases involved in the global TS-SQL query will be determined. Some query optimization heuristics are introduced to reduce the processing overhead. Similar strategies have been adopted for optimizing queries for other multidatabase systems [7,8]. Details of query decomposition will be given in Section 7.

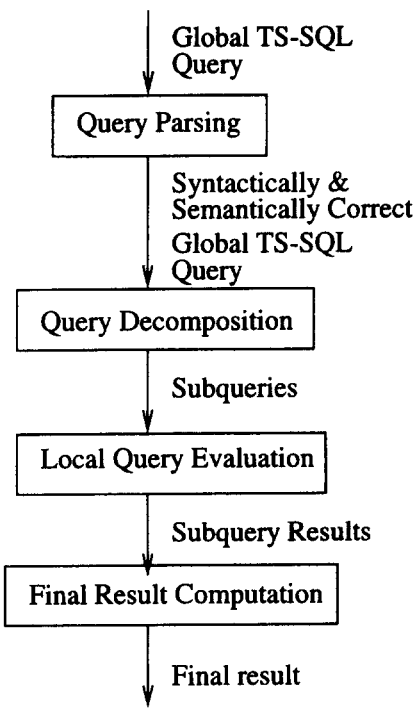



Fig. 4. Distributed query processing steps.

(3) **Local query evaluation.** Decomposed subqueries are disseminated to the appropriate query agents for execution. Query agents further translate the subqueries into local database queries and return the subquery results to the query mediator.
(4) **Final result computation.** The query mediator assembles the subquery results and computes the final query result if there remain some query operations that could not be performed by the query agents. Examples of such query operations are attribute projection and aggregate functions in a SELECT clause with ANY_DB source option.

## 7. Query decomposition with optimization

Unlike the traditional SQL queries, TS-SQL allows source options to be attached to their SELECT and WHERE clauses. Hence, our query decomposition strategy has to handle TS-SQL queries with different combination of source options. In decomposing global queries, our query decomposition strategy is designed to fulfil the following objectives:

- As far as possible, we would like the query agents to perform most query processing tasks in order to maximize the parallelism in local query evaluation.
- Heuristic query optimization has to be performed to reduce the subquery results as well as the local query results that have to be transferred from the local database sites to the query mediator. With small local query results shipped between sites, we can improve the query response time.
- Since not all TS-SQL operations can be performed by the local database systems, the decomposition process must consider the capabilities of query agents and also determine the portion(s) of global queries to be processed by the query mediator itself. At present, we have assumed that all query agents support the usual SELECT-PROJECT-JOIN SQL queries.

The source option on a SELECT clause determines if the source values of tuples have to be merged during the projection operation. On the other hand, the source option on a WHERE clause that involves more than one TS-relations determines how the join between the TS-relations is performed. In other words, if the source option SAME_DB is assigned to a WHERE clause, only tuples from the same local databases are allowed to be joined together. If the source option ANY_DB is assigned to a WHERE clause, tuples from any local database can be joined together.

By exploiting this join definition, we derive the decomposition strategies for the following two categories of TS-SQL queries:

- TS-SQL queries with SAME_DB assigned to their WHERE clauses, and
- TS-SQL queries with ANY_DB assigned to their WHERE clauses.

### *7.1. Decomposition strategy for TS-SQL queries with 'WHERE · · · SAME_DB'*

Given a global query in this category, we decompose it into a **subquery template** and a **global query residue** as shown in Fig. 5a. Here, the subquery template is a subquery generated based on the global schema and it has to be further translated into subqueries on the export schemas of local databases relevant to the global query. The global query residue represents the remaining global query operations that have to be handled by the query mediator.

Since SAME_DB is the source option of the WHERE clause, all selection and join predicates on the global TS-relation(s) can be performed by the query agents together with their local database

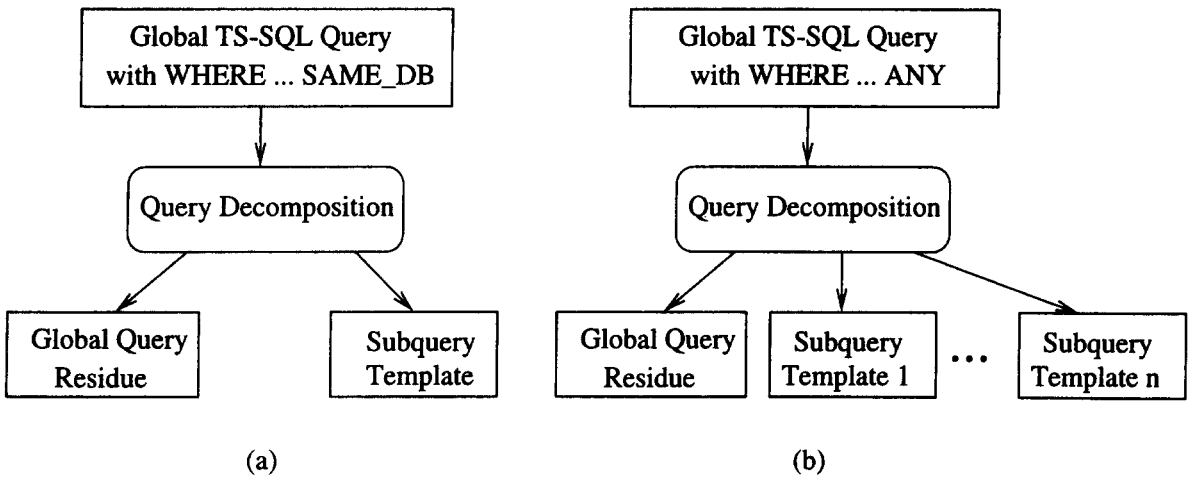



Fig. 5. Distributed query processing steps.

systems. For example, in the following query *Qa*, we show that the join predicate ($E1.dept = D1.dname$) and selection predicate ($E1.salary > 3000$) have been propagated to the subquery template. Having performed a union of subquery results returned by the query agents, a final projection operation on the union result will be required as specified in the global query residue.

Global Query (Qa):
SELECT E1.ename, D1.manager [ANY_DB]
FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname AND E1.salary > 3000 [SAME_DB]
Subquery Template:
SELECT E1.ename, D1.manager
FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname AND E1.salary > 3000
Global Query Residue:
SELECT E1.ename, D1.manager [ANY_DB]
FROM ⟨union of subquery results⟩

As shown in the above example, the subquery template involves global relation names and global attribute names. It has to be further translated into subqueries on the export schemas. In the following, we briefly describe the steps of deriving the subquery template and global query residue from a global query:

(1) The SELECT clause of the subquery template is assigned the list of attributes that appears in the SELECT clause of the global query, including those which appear in the aggregate functions.[2]
(2) The FROM clause of the subquery template is assigned the global relations that appear in the FROM clause of the global query.

[2] The decomposition can be further optimized by having the aggregate functions assigned to the subquery template when the SAME_DB option is attached to the SELECT clause of the global query.

(3) Move the selection and join predicates in the WHERE clause of global query to the WHERE clause of the subquery template.
(4) The global query residue inherits the SELECT clause of the original global query. Its FROM clause is defined by a union of subquery results. In other words, the only operations to be performed by the global query residue are projections, aggregations and groupbys.

### *7.2. Decomposition strategy for TS-SQL queries with 'WHERE ... ANY_DB'*

For a global query in this category, our decomposition strategy generates a global query residue and multiple subquery templates, one for each global relation involved in the global query. This is shown in Fig. 5b. In other words, a global query with $n$ relations in its FROM clause will be decomposed into $n$ subquery templates. This is necessary because join predicates in the global query cannot be propagated to the subqueries.

For example, in the following query *Qb*, it can be shown that the join predicate ($E1.dept = D1.dname$) cannot be evaluated before the two global TS-relations *Emp* and *Dept* are derived. Nevertheless, the selection predicate ($E1.salary > 3000$) can still be propagated to the subqueries for the local relations corresponding to *Emp*. Having performed unions of subquery results to construct the global relations *Emp* and *Dept*, a final join and projection on the global relations will be required as specified in the global query residue.

Global Query (Qb):
SELECT E1.ename, D1.manager [SAME_DB]
FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname AND E1.salary > 3000 [ANY_DB]
Subquery Template 1 (for *Emp*):
SELECT E1.ename, E1.dept FROM Emp E1
WHERE E1.salary > 3000
Subquery Template 2 (for *Dept*):
SELECT D1.manager, D1.dname FROM Dept D1
Global Query Residue:
SELECT E1.ename, D1.manager [SAME_DB]
FROM ⟨union of subquery results for *Emp* > E1,
⟨union of subquery results for *Dept* > D1
WHERE E1.dept = D1.dname [ANY_DB]

The steps of deriving the subquery templates and global query residue from a global query are:

(1) For each global TS-relation ($R$) involved in the FROM clause, we generate its corresponding subquery template as follows:
   (a) The SELECT clause of the subquery template is assigned the list of $R$'s attributes that appears in the SELECT clause of the global query, including those that appear in the aggregate functions.
   (b) Selection and join predicates using $R$'s attributes in the global query are propagated to the subquery template.
   (c) The FROM clause of the subquery template is assigned $R$.
(2) The inter-global relation join predicates in the WHERE clause of global query, the projection and aggregation are retained in the WHERE clause of the global query residue.

### 7.3. Translation of subquery templates into subqueries

In this section, we describe how the subquery templates decomposed from the global queries are translated into subqueries for the query agents. Although the query agents support subqueries on the export schemas which are compatible to the global schema, translating subquery templates into subqueries is still necessary for the following reasons:

- **Missing export relations.** In our multidatabase query processor design, a global relation may not always have its corresponding export relation in every export schema. When a global relation in the FROM clause of a subquery template cannot be found in an export schema, no subquery will be generated for the corresponding query agent.
- **Missing attributes.** Some global attributes may not be found in the export schema. If any global attribute involved in a WHERE clause of a subquery template cannot be found in the export schema defined upon a local database, it is not required to translate the subquery template for the query agent of the local database. However, if the WHERE clause involves an explicit check for NULL, a translated subquery is still required for each query agent.

## 8. Conclusions

In this paper, we extend the relational model to accommodate instances from multiple databases while keeping their source information. The new model, called TS relational model, is designed based on the scenario that a multidatabase consists of an integrated schema and instances which are not integrated. We believe that it is not always desirable to perform complete instance integration as there are global users/applications which just require a global schema to uniformly query the local databases. It is important for these users/applications to identify the source of the instances in order to make decisions. To query the TS relations, we propose a full set of tuple-source relational operations and extend the SQL language. The correctness of these TS relational operations has been established by showing that they are consistent with the export relational operations. Based on our architecture design, a distributed query processor supporting TS-SQL queries over multiple local databases has been implemented on the UNIX platform. In our implementation, a public domain database package known as Mini-SQL [10] has been used for managing local databases. Interaction between query mediator and its query agents have been realized using UNIX message queues.

The future work includes:

- *User friendly query interface.* We are now in the process of designing and implementing a user-friendly query interface for TS-SQL queries in the heterogeneous database environment. The query interface is expected to implemented on the web so that TS-SQL could be widely applied to public domain databases.
- *Queries to non-traditional databases.* Although TS-SQL has been originally designed for querying distributed structured databases, we are looking forward to extending it to query widely available and possibly semi- or un-structured data on the Internet.
- *Data model and query operations for integrated databases with level-1 instance integration.* In this paper, we describe the extended relational model in the context of no instance integration performed on the export relations. The future work will explore an appropriate extension to the

relational model for integrated databases with level-1 instance integration. In such integrated databases, entity identification is performed but not attribute value conflict resolution. Hence, it is necessary to attach source values to the attribute values instead of the tuples.

## Appendix A. Formal properties of TS relational model

In this appendix, we describe some formal properties about the TS relational model. In Section A.1, we establish a mapping between TS relational operations and the relational operations on the underlying export databases. The one-to-one mapping allows us to demonstrate the consistency of TS relational model with respect to the relational operations on the export databases. For query processing purposes, knowledge about the algebraic properties of TS relational model is required. In Section A.2, we present a set of algebraic rules that can be used to optimize TS relational queries.

### A.1. Correctness of the TS relational model

Although TS relational model is designed for representing global relations, it is closely related to the component export databases due to the existence of source attribute. It is therefore appropriate to characterize the correctness of TS relational model by showing that the TS relational operations are consistent with the relational operations on the export relations. Henceforth, we call these relational operations the **export relational operations**. Nevertheless, for TS relational operations that combine tuples from different export databases, i.e. $\overset{s}{\pi}^{any}$, $\overset{s}{\bowtie}^{any}$, $\overset{s}{\cup}^{any}$, $\overset{s}{\cap}^{any}$ $\overset{s}{-}^{any}$, $\overset{s}{aggreg}^{any}$ and $\overset{s}{groupby}^{any}$, it is not possible to show their consistency with respect to the export relational operations because these TS relational operations offer query expressiveness beyond that of the export relational operations. Hence, in the following, we will only demonstrate the correctness of TS relational operations which can be mapped into export relational operations.

**Definition (Consistency of Global Operation).** Given $n$ export databases (say $DB_1, \ldots, DB_n$), and a merging operation (denoted by *merge*( )) that combines the export relations into global relations, a global relational operation (denoted by $OP_G$) is consistent with the export relational operations (denoted by $OP_L$) if and only if there exists a 1-1 mapping (denoted by *opMap*) from $OP_G$ to $\langle (OP_L \cup \{op_\phi\}), \ldots, (OP_L \cup \{op_\phi\})\rangle$[3] such that $\forall L_{ij} \in Rel(DB_i)$, $\forall op_G \in OP_G$ (say, $op_G$ is $m$-ary),

$$op_G(G_1, \ldots, G_m) = merge(op_{L_1}(L_{11}, \ldots, L_{n1}), \ldots, op_{Lm}(L_{1m}, \ldots, L_{nm}))$$

where $G_i = merge(L_{1i}, L_{21}, \ldots, L_{ni})$

(i.e. global relation $G_i$ is derived by combining export relations $L_{1i}, \ldots, L_{ni}$ from $DB_1, \ldots, DB_n$ respectively. Without loss of generality, we assume that any $DB_k$ that has no export relation for deriving $G_i$ will have $L_{ki} = \phi$), and $opMap(op_G) = \langle op_{L_1}, \ldots, op_{L_m}\rangle$. □

In the following, we show an 1-1 operation mapping function from global relational operations to export relational operations:

[3] $op_\phi$ is an operation that returns an empty relation for any given input–export relation.

$opMap(\overset{s}{\sigma}_{p \wedge (source \in DBSet)}) = \langle op_1, \ldots, op_n \rangle$ where:

$$op_i = \begin{cases} \sigma_p & DB_i \in DBSet \\ op_\phi & \text{otherwise} \end{cases}$$

$opMap(\overset{s}{\sigma}_p) = \langle \sigma_p, \ldots, \sigma_p \rangle$

$opMap(\overset{s}{\pi}_a^{same\_DB}) = \langle \pi_a, \ldots, \pi_a \rangle$

$opMap(\overset{s}{\bowtie}_p^{same\_DB}) = \langle \bowtie_p, \ldots, \bowtie_p \rangle$

$opMap(\overset{s}{\cup}^{same\_DB}) = \langle \cup, \ldots, \cup \rangle$

$opMap(\overset{s}{\cap}^{same\_DB}) = \langle \cap, \ldots, \cap \rangle$

$opMap(\overset{s}{-}^{same\_DB}) = \langle -, \ldots, - \rangle$

$opMap(\overset{s}{aggreg}^{same\_DB}) = \langle aggreg, \ldots, aggreg \rangle$

$opMap(\overset{s}{groupby}^{same\_DB}) = \langle groupby, \ldots, groupby \rangle$

The above operation mapping function will be used to prove the consistency of the TS relational operations annotated by *same_DB*.

**Proofs of consistent global operations.** The following two lemmas show that global TS selection is consistent with or without source predicates.

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}$, and $G_j = merge(L_{1j}, \ldots, L_{nj})$, $\overset{s}{\sigma}_{p \wedge (source \in DBSet)} G_j = merge(op_1 L_{1j}, \ldots, op_n L_{nj})$ *where*

$$op_i = \begin{cases} \sigma_p & DB_i \in DBSet \\ op_\phi & \text{otherwise.} \end{cases}$$

**Proof.**

$g \in \overset{s}{\sigma}_{p \wedge (source \in DBSet)} G_j \Leftrightarrow$

$g \in G_j$, $g$ satisfies $p$, and $\exists DB_i \in DBSet$, $g.source = DB_i \Leftrightarrow$

$\exists DB_i \in DBSet$, $g.A \in \sigma_p L_{ij}$ where $A = Attr(G) = Attr(L_{ij}) \Leftrightarrow$

$g \in merge(op_1 L_{1j}, \ldots, op_{i-1} L_{(i-1)j}, op_i L_{ij}, op_{i+1} L_{(i+1)j}, \ldots, op_n L_{nj})$ where

$$op_i = \begin{cases} \sigma_p & DB_i \in DBSet \\ op_\phi & \text{otherwise.} \end{cases}$$ □

In the above proof, the source predicate constrains the result of global selection to contain only tuples with $DB_i$ as source values. This implies that the tuple attributes come from an export relation in $DB_i$. Hence a global selection operation produces result identical to that produced by first performing a selection on the export relation from $DB_i$ followed by making the tuples global using the *merge* operation.

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}$, *and* $G_j = merge(L_{1j}, \ldots, L_{nj})$, $\overset{s}{\sigma}_p G_j = merge(\sigma_p L_{1j}, \ldots, \sigma_p L_{nj})$.

**Proof.** $g \in \overset{s}{\sigma}_p G_j \Leftrightarrow \exists i \in \{1, \ldots, n\}$, $g.A \in \sigma_p L_{ij}$ where $A = Attr(G) = Attr(L_{ij}) \Leftrightarrow$
$g \in merge(\sigma_p L_{1j}, \ldots, \sigma_p L_{ij}, \ldots, \sigma_p L_{nj})$. □

As shown below, every global projection operation with *same_DB* option is equivalent to projecting the required attributes from the export relations first followed by merging the projected export relations. Hence the global TS project operation is consistent.

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}$, and $G_j = merge(L_{1j}, \ldots, L_{nj})$, $\overset{s}{\pi}_A^{same\_DB} G_j = merge(\pi_A L_{1j}, \ldots, \pi_A L_{nj})$.

**Proof.**

$g \in \overset{s}{\pi}_A^{same\_DB} G_j$ and $g.source = DB_i$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$
$g.A \in \pi_A L_{ij}$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$
$g = (g.A, DB_i)$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$
$g \in merge(\pi_A L_{1j}, \ldots, \pi_A L_{nj})$ □

Using similar proof techniques, we can also show that global TS join, union, intersection, aggregation and groupby operations are consistent with the respective relational operations on the export relations.

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}, L_{1k}, \ldots, L_{nk}$, $G_j = merge(L_{1j}, \ldots, L_{nj})$, *and* $G_k = merge(L_{1k}, \ldots, L_{nk})$
$G_j \overset{s}{\bowtie}_p^{same\_DB} G_k = merge(L_{1j} \bowtie_p L_{1k}, \ldots, L_{nj} \bowtie_p L_{nk})$.

**Proof.**

$g \in (G_j \overset{s}{\bowtie}_p^{same\_DB} G_k)$ and $g.source = DB_i$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$
$((g.A_j, DB_i) \in G_j)$ and $((g.A_k, DB_i) \in G_k)$ and $(g.A_j, g.A_k$ satisfy the join predicate $p)$ for some $i \Leftrightarrow$
$(g.A_j \in L_{ij})$ and $(g.A_k \in L_{ik})$ and $(g.A_j, g.A_k$ satisfy $p)$ for some $i \Leftrightarrow$
$(g.A_j, g.A_k) \in (L_{ij} \bowtie_p L_{ik})$ for some $i \Leftrightarrow$
$g = (g.A_i, g.A_k, DB_i)$ for some $i \Leftrightarrow$
$g \in merge(L_{1j} \bowtie_p L_{1k}, \ldots, L_{nj} \bowtie_p L_{nk})$ □

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}, L_{1k}, \ldots, L_{nk}$, $G_j = merge(L_{1j}, \ldots, L_{nj})$, and $G_k = merge(L_{1k}, \ldots, L_{nk})$
$G_j \overset{s}{\cup}^{same\_DB} G_k = merge(L_{1j} \cup L_{1k}, \ldots, L_{nj} \cup L_{nk})$

**Proof.**

$g \in (G_j \overset{s}{\cup}^{same\_DB} G_k) \Leftrightarrow$
$(g \in G_j) \vee (g \in G_k)$ and $g.source = DB_i$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$
$(g.A \in L_{ij}) \vee (g.A \in L_{ik})$ for some $i \Leftrightarrow$
$g.A \in (L_{ij} \cap L_{ik})$ for some $i \Leftrightarrow$
$g \in merge(L_{1j} \cup L_{1k}, \ldots, L_{nj} \cup L_{nk})$ □

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}, L_{1k}, \ldots, L_{nk}$, $G_j = merge(L_{1j}, \ldots, L_{nj})$ and $G_k = merge(L_{1k}, \ldots, L_{nk})$
$G_j \overset{s}{\cap}^{same\_DB} G_k = merge(L_{1j} \cap L_{1k}, \ldots, L_{nj} \cap L_{nk})$.

**Proof.**

$g \in (G_j \overset{s}{\cap}^{same\_DB} G_k) \Leftrightarrow$
$(g \in G_j) \wedge (g \in G_k)$ and $g.source = DB_i$ for some $i \in \{1, \ldots, n\} \Leftrightarrow$

$(g.A \in L_{ij}) \wedge (g.A \in L_{ik})$ for some $i \Leftrightarrow$
$g \in merge(L_{1j} \cap L_{1k}, \ldots, L_{nj} \cap L_{nk})$. □

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}, L_{1k}, \ldots, L_{nk}$, $G_j = merge(L_{1j}, \ldots, L_{nj})$, and $G_k = merge(L_{1k}, \ldots, L_{nk})$
$G_j \overset{s}{-}^{same\_DB} G_k = merge(L_{1j} - L_{1k}, \ldots, L_{nj} - L_{nk})$.

**Proof.**
$g \in (G_j \overset{s}{-}^{same\_DB} G_k) \Leftrightarrow$
$(g \in G_j) \wedge (g \notin G_k)$ and $\exists i$, $g.source = DB_i \Leftrightarrow$
$(g.A \in L_{ij}) \wedge (g.A \notin L_{ik})$ for some $i \Leftrightarrow$
$g.A \in (L_{ij} - L_{ik})$ for some $i \Leftrightarrow$
$g \in merge(L_{1j} - L_{1k}, \ldots, L_{nj} - L_{nk})$. □

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}$, $G_j = merge(L_{1j}, \ldots, L_{nj}) \overset{s}{aggreg}^{same\_DB} G_j(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle) =$
$merge((aggreg\ L_{1j}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)), \ldots, (aggreg\ L_{nj}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)))$.

**Proof.**
$g \in \overset{s}{aggreg}^{same\_DB} G_j(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle) \Leftrightarrow$
$g = (x_1, \ldots, x_m, DB_i)$ for some $i \in \{1, \ldots, n\}$ where $x_1 = f_1(\pi_{A_1}(\sigma_{source = DB_i} G_j)) \wedge \cdots \wedge x_m =$
$f_m(\pi_{A_m}(\sigma_{source = DB_i} G_j)) \Leftrightarrow$
$x_1 = f_1(\pi_{A_1} L_{ij}) \wedge \cdots \wedge x_m = f_m(\pi_{A_m} L_{ij})$ for some $i \Leftrightarrow$
$g = (x_1, \ldots, x_m, DB_i) \in merge((aggreg\ L_{1j}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)), \ldots,$
$(aggreg\ L_{nj}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)))$. □

**Lemma.** $\forall L_{1j}, \ldots, L_{nj}$, $G_j = merge(L_{1j}, \ldots, L_{nj})$ $\overset{s}{groupby}^{same\_DB} G_j(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B) =$
$merge((groupby\ L_{1j}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B)), \ldots, (groupby\ L_{nj}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B)))$.

**Proof.**
$g \in \overset{s}{groupby}^{same\_DB} G_j(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B)$. Let $g.B = b$ and $g.source = DB_i$ for some $i \Leftrightarrow$
$b \in \pi_B(\sigma_{source = DB_i} G_j)$, and $g.A = (f_1(\pi_{A_1}(\sigma_{(B=b) \wedge (source = DB_i)} G_j)), \ldots,$
$f_m(\pi_{A_m}(\sigma_{(B=b) \wedge (source = DB_i)} G_j)), b)$ where $A = Attr(G_j) - \{source\} \Leftrightarrow$
$b \in \pi_B L_{ij}$ and $g.A = (f_1(\pi_{A_1}(\sigma_{(B=b)} L_{ij})), \ldots, f_m(\pi_{A_m}(\sigma_{(B=b)} L_{ij})), b) \Leftrightarrow$
$g.A \in groupby\ L_{ij}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B) \Leftrightarrow$
$g \in merge((groupby\ L_{1j}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle))(B), \ldots,$
$(groupby\ L_{nj}(\langle f_1, A_1\rangle, \ldots, \langle f_m, A_m\rangle)(B)))$ □

Having proved the consistency of the above TS operations, the following corollary becomes apparent.

**Corollary.** *The set of TS operations* $\{\overset{s}{\sigma}, \overset{s}{\pi}^{same\_DB}, \overset{s}{\bowtie}^{same\_DB}, \overset{s}{\cup}^{same\_DB}, \overset{s}{\cap}^{same\_DB}, \overset{s}{-}^{same\_DB}, \overset{s}{aggreg}^{same\_DB}, \overset{s}{groupby}^{same\_DB}\}$, is consistent with respect to the export relational operations.

## A.2. Algebraic properties of the TS relational model

Given a global multidatabase query written in TS-SQL, a distributed query processor derives its equivalent TS relational expression and evaluates the TS relational operations in the expression. As part of the query evaluation process, the query processor may have to transform the relational expression in order to obtain an algebraically equivalent relational expression that requires the least evaluation cost (in terms of disk and communication overheads). Such transformation of TS queries can only be possible when the algebraic properties of TS relational model are known. In this section, we will present a few important algebraic properties related to the join operation in the TS relational model.

**Theorem.** $\overset{s}{\bowtie}$ *is commutative.*

**Proof.** This can be easily shown as the definition does not depend on the ordering of operands. While the commutativity of $\overset{s}{\bowtie}$ can be proven easily, the associativity of $\overset{s}{\bowtie}$ has to be shown by considering $\overset{s}{\bowtie}^{same\_DB}$ and $\overset{s}{\bowtie}^{any}$ separately.

**Theorem.** $\overset{s}{\bowtie}^{same\_DB}$ is associative, i.e.
$(R \overset{s}{\bowtie}^{same\_DB} S) \overset{s}{\bowtie}^{same\_DB} T = R \overset{s}{\bowtie}^{same\_DB} (S \overset{s}{\bowtie}^{same\_DB} T)$.

**Proof.** Note that every tuple in the $\overset{s}{\bowtie}^{same\_DB}$ result must have non-* source attribute value. Hence,

$(a_r, a_s, a_t, s) \in (R \overset{s}{\bowtie}^{same\_DB} S) \overset{s}{\bowtie}^{same\_DB} T \Leftrightarrow$
$(a_r, a_s, s) \in R \overset{s}{\bowtie}^{same\_DB} S$ and $(a_t, s) \in T$ and they satisfy the join predicate $\Leftrightarrow$
$(a_r, s) \in R$, $(a_s, s) \in S$ and $(a_t, s) \in T$ and they satisfy the join predicate $\Leftrightarrow$
$(a_r, s) \in R$, and$(a_s, a_t, s) \in (S \overset{s}{\bowtie}^{same_DB} T)$ and they satisfy the join predicate $\Leftrightarrow$
$(a_r, a_s, a_t, s) \in R \overset{s}{\bowtie}^{same\_DB} (S \overset{s}{\bowtie}^{same\_DB} T)$. □

**Theorem.** $\overset{s}{\bowtie}^{any}$ *is associative, i.e.* $(R \overset{s}{\bowtie}^{any} S) \overset{s}{\bowtie}^{any} T = R \overset{s}{\bowtie}^{any} (S \overset{s}{\bowtie}^{any} T)$.

**Proof.** Every tuple in the $\overset{s}{\bowtie}^{any}$ result can either have non-* or * source attribute value.

*Case 1 (result tuple source attribute is non-*, say s).* It can be shown using the procedure similar to the previous proof that: $(a_r, a_s, a_t, s) \in (R \overset{s}{\bowtie}^{any} S) \overset{s}{\bowtie}^{any} T \Leftrightarrow (a_r, a_s, a_t, s) \in R \overset{s}{\bowtie}^{any} (S \overset{s}{\bowtie}^{any} T)$

*Case 2 (result tuple source attribute is *).* $(a_r, a_s, a_t, *) \in (R \overset{s}{\bowtie}^{any} S) \overset{s}{\bowtie}^{any} T \Leftrightarrow (a_r, s_r) \in R$ and $(a_s, s_s) \in S$ and $(a_t, s_t) \in T$ and they satisfy the join predicate. In other words, $(\exists s_i \in \{s_r, s_s, s_t\}, \quad (s_i = *)) \vee (\exists s_i, s_j \in \{s_r, s_s, s_t\}, \quad s_i \neq s_j) \Leftrightarrow (a_r, a_s, a_t, *) \in R \overset{s}{\bowtie}^{any} (S \overset{s}{\bowtie}^{any} T)$. Hence, $(R \overset{s}{\bowtie}^{any} S) \overset{s}{\bowtie}^{any} T = R \overset{s}{\bowtie}^{any} (S \overset{s}{\bowtie}^{any} T)$. □

Given the above theorems, one can now choose any join evaluation sequence for a TS relational expression as long as the join operations are of the same type, i.e. either *same_DB* or *any*. Although

$\overset{s}{\bowtie}^{same\_DB}$ and $\overset{s}{\bowtie}^{any}$ are associative when they are not mixed, they are mutually non-associative as shown in the following theorem.

**Theorem.** $(R \overset{s}{\bowtie}^{same\_DB} S) \overset{s}{\bowtie}^{any} T \neq R \overset{s}{\bowtie}^{same\_DB} (S \overset{s}{\bowtie}^{any} T)$.

**Proof.** This can be shown by the following counter-example:

Relation $R$

| a | Source |
|---|---|
| 1 | $DB_A$ |

Relation $S$

| b | Source |
|---|---|
| 1 | $DB_A$ |

Relation $T$

| c | Source |
|---|---|
| 1 | $DB_B$ |

Relation $(R \overset{s}{\bowtie}_{a=b}^{same\_DB} S) \overset{s}{\bowtie}_{b=c}^{any} T$

| *a* | *b* | *c* | *Source* |
|---|---|---|---|
| 1 | 1 | 1 | * |

However, $(R \overset{s}{\bowtie}_{a=b}^{same\_DB} (S \overset{s}{\bowtie}_{b=c}^{any} T)$ is empty. □

While mixed associativity does not hold for TS join operations, it does not pose any problem to the evaluation of TS-SQL queries since the present TS-SQL syntax does not allow mixed joins.

## References


[1] S. Agarwal, A.M. Keller, G. Wiederhold, K. Saraswat. Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In: *Proc. Intl. Conf. on Data Engineering* (1995) 495–504.

[2] C. Batini, M. Lenzerini, S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys* 18(4) (December 1986) 323–364.

[3] R.H.L. Chiang, T.M. Barron, V.C. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data and Knowledge Engineering* 12(2) (1994) 107–142.

[4] D. Clements, M. Ganesh, S.-Y. Hwang, E.-P. Lim, K. Mediratta, J. Srivastava, J. Stenoein, H. Yang. Myriad: Design and implementation of a federated database prototype. In: *Proc. ACM SIGMOD Conf.* (1994) 518.

[5] A. Chatterjee, A. Segev. Data manipulation in heterogeneous databases. *SIGMOD Record* 20(4) (December 1991) 64–68.

[6] L.G. DeMichiel. Resolving database incompatibility: an approach to performing relational operations over mismatched domains. *IEEE Trans. on Knowledge and Data Engineering* 1(4) (1989) 485–493.

[7] C. Evrendilek, A. Dogac, S. Nural, F. Ozcan. Query optimization in multidatabase systems, *Journal of Distributed and Parallel Databases* 5(1) (1997) 77–114.

[8] B. Finance, V. Smahi, J. Fessy. Query processing in IRO-DB. In: *Proc. 4th Intl. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*, Singapore (December 1995) 299–318.

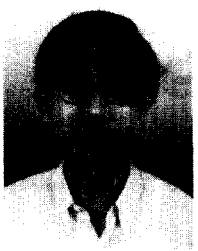
[9] C. Fahrner, G. Vossen. A survey of database design transformations based on the entity-relationship model. *Data and Knowledge Engineering* 15(3) (June 1995) 213–250.

[10] D.J. Hughes. Mini SQL, Version 1.0, *Minerva Network Management Environment* (1990).

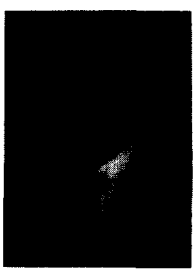
[11] S. Hayne, S. Ram. Multi-user view integration system (MUVIS): An expert system for view integration. In: *Proc. Intl. Conf. on Data Engineering* (1990) 402–409.

[12] M. Kaul, K. Drosten, E.J. Neuhold. Integrating heterogeneous information bases by object-oriented views. In: *Proc. Intl. Conf. on Data Engineering* (1990) 2–10.

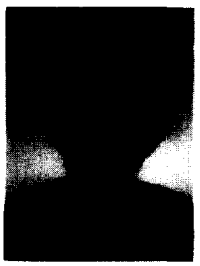
[13] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas. MSQL: A multidatabase language, *Information Sciences* 49 (1989) 59–101.
[14] J.A. Larson, S.B. Navathe, R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. on Software Engineering* 15(4) (April 1989) 449–463.
[15] L. Liu, C. Pu, Y. Lee. An adaptive approach to query mediation across heterogeneous databases. In: *Proc. Intl. Conf. on Cooperative Information Systems* (June 1996) 144–156.
[16] E.-P. Lim, J. Srivastava, S. Prabhakar, J. Richardson. Entity identification problem in database integration. In: *Proc. Intl. Conf. on Data Engineering* (1993) 294–301.
[17] E.-P. Lim, J. Srivastava, S. Shekhar. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In: *Proc. Intl. Conf. on Data Engineering* (1994) 154–163.
[18] L.V.S. Lakshmanan, F. Sadri, I.N. Subramanian. Schema SQL—A language for interoperability in relational multi-database systems. In: *Proc. Intl. Conf. on Very Large Data Bases* (1996) 239–250.
[19] A. Meier et al., Hierarchical to relational database migration. *IEEE Software* (May 1994) 21–27.
[20] M.P. Reddy, B.E. Prasad, P.G. Reddy, A. Gupta. A methodology for integration of heterogeneous databases. *IEEE Trans. on Knowledge and Data Engineering* 6(6) (December 1994) 920–933.
[21] F. Saltor, M. Castellanos, M. Garcia-Solaco. Suitability of data models as canonical models for federated databases. *SIGMOD Record* 20(4) (December 1991) 44–48.
[22] A.P. Sheth, J.A. Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. *ACM Computing Surveys* 22(3) (September 1990) 183–236.
[23] S. Spaccapietra, C. Parent, Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *Very Large Database Journal* 1(1) (1992) 81–126.
[24] P.S.M. Tsai, A.L.P. Chen. Querying uncertain data in heterogeneous databases. In: *Proc. of RIDE-IMS Conf.* (1993) 161–168.
[25] Y.R. Wang, S.E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In: *Proc. Intl. Conf. on Data Engineering* (1989) 46–55.
[26] R. Wang, S. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In: *Proc. Intl. Conf. on Very Large Data Bases* (1990) 519–538.
[27] C. Zaniolo. Design of relational views over network schemas. In: *Proc. ACM SIGMOD Intl. Conf. on Management of Data* (1979) 179–190.
[28] J.L. Zhao, A. Segev, A. Chatterjee. A universal relation approach to federated database management. In: *Proc. Interl. Conf. on Data Engineering* (1995) 261–270.



**Ee-Peng Lim** received the B.S. (Hons.) degree in Information Systems and Computer Science from the National University of Singapore, in 1989, and the Ph.D. degree in Computer Science from the University of Minnesota, Minneapolis, in 1994. Since 1994, he has been on the faculty of the School of Applied Science at the Nanyang Technological University, Singapore, where he is currently a lecturer and also the Director of the Centre for Advanced Information Systems (CAIS). His current research interests include web computing, multi-database systems, and digital libraries. He has published more than 35 papers in journals, conferences and workshops. He is also the principal investigator of HARP, an integrated digital library project jointly supported by Nanyang Technological University and National Computer Board, Singapore. Dr. Lim is a member of the IEEE Computer Society and the ACM. His professional activities have included being on the program committees of the IEEE Conference on Tools of Artificial Intelligence 1995 (USA) and the ACM Digital Library Conference 1997 (USA), and refereeing for journals and conferences.



**Roger H.L. Chiang** is a Senior Lecturer of the Division of Strategy and Information and Technology, Deputy Director of the Information Management Research Centre, and Deputy Programme Director of MBA in Management of Information Technology, Nanyang Business School, Nanyang Technological University, Singapore. He has a B.S. degree in Management Science from National Chiao Tung University, Taiwan, M.S. degrees in Computer Science from Michigan State University and in Business Administration from University of Rochester, and a Ph.D. degree in Computers and Information Systems from University of Rochester. His Ph.D. dissertation focuses on developing a methodology for database reverse engineering. He was a finalist in the 1993 Ph.D. dissertation competition of the International Conference on Information Systems. His research interests are in database and expert systems, particularly in database reverse engineering, database integration, and common sense reasoning and learning. His research has been published in a number of international journals including ACM Transactions on Database Systems, Data and Knowledge Engineering, Decision Support Systems, and the Journal of Database Administration. He is a member of AAAI, ACM, AIS, IEEE and INFORMS.



**Yinyan Cao** graduated from the Nanyang Technological University in 1997 with a first-class honours degree in Computer Engineering. Her research interests include multidatabases and resource discovery. She is currently working at the Centre for Advanced Information Systems, Nanyang Technological University as a project officer.